VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Systém pro správu překladů

Translation Management System

Jiří Hájíček

Bakalářská práce

Vedoucí práce: Ing. Jan Janoušek

Ostrava, 2021

Abstrakt

Cílem této bakalářské práce je vytvoření funkční webové aplikace pro správu překladů. V první části této práce budou představena různá existující řešení v dané oblasti, budou popsány jejich hlavní funkce, které nabízí. V druhé části práce budou představeny moderní technologii pro tvorbu webových aplikací, jejich hlavní výhody, funkce a vlastnosti, které nabízí. V poslední části bude popsán návrh a implementace řešení pro systém pro správu překladů. Na konci bude shrnuta dosažena aplikace s jejími výsledky.

Klíčová slova

správa překladů, lokalizace, ReactJS, webová aplikace

Abstract

The aim of this bachelor thesis is to create a functional web application for translation management. In the first part of this work, various existing solutions in the field will be introduced, their main functions will be described. The second part of the work will introduce modern technologies for creating web applications, their main advantages, functions and features it offers. The last part will present the design and implementation of a solution for a translation management system. At the end, a whirlwind will be achieved by the application with its results.

Keywords

translation management, localization, ReactJS, web application

Poděkování

Rád bych na tomto místě poděkoval vedoucímu této práce Ing. Janu Janouškovi, především za jeho odborné vedení a za cenné a praktické rady při zpracování této bakalářské práce.

Obsah

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XML
API	–	Application Programming Interface
CD	–	Continuous Deployment
CI	–	Continuous Integration
CLI	–	Command Line Interface
CSS	–	Cascading Style Sheets
DOM	–	Document Object Model
HTML	–	Hypertext Markup Language
JS	–	JavaScript
JSON	–	JavaScript Object Notation
JSX	–	JavaScript XML
MVC	–	Model-view-controller
PHP	–	Hypertext Preprocessor
SDK	–	Software development kit
UI	–	User interface
URL	–	Uniform Resource Locator

Seznam obrázků

Kapitola 1

Úvod

V dnešní době je kladen velký důraz na lokalizaci aplikací, u které je očekávána správná gramatika i správný význam přeložené hlášky či věty. U menších, jednodušších projektů je možné překlady uchovávat v jednom souboru uložené pod klíčem, avšak u větších projektů se stovky až tisíci překlady může být takové řešení dost složité, překlady nemusí být správné a editace takovýchto souboru vzhledem k množství textu může být dost náročná. Proto existují řešení, která se vývojářům aplikací snaží tyto problémy vyřešit a zlepšit kvalitu překladů a usnadnit uživatelům, kteří překlady spravují, jejich editaci a další práci s překlady.

Cílem této práce je představení různých existujících řešení v dané oblasti, která se v tuto chvíli nacházejí na trhu. Každé řešení, které bude popsáno, bude obsahovat soupis funkcí a vlastností nabízené daným řešením.

Dalším výstupem bakalářské práce je představení moderních technologií pro tvorbu webových aplikací. Práce bude zaměřena hlavně na moderní nejčastěji používané JS frameworky, které se v tuto chvíli hojně využívají při tvorbě webových aplikací.

Hlavní cíl této práce je však vytvoření webové aplikace, která bude sloužit jako systém pro správu překladů. Tato aplikace by měla využívat web API třetích stran, k zjednodušení práce uživatele při překládání a následné kontrole překladů. Tato práce by měla tedy obsahovat i popis tohoto programu a všech jeho vlastností, které do této aplikace byly implementovány.

Kapitola 2

Přehled existujících řešení

Systémů pro správu překladů je na dnešním trhu nespočet, i když u menších projektů nemusí být použití takového sytému pro lokalizaci nutné, obsahují tyto systémy spoustu funkcí, které uživatel ocení, a které mu usnadní práci. Hlavní výhody, které tyto systémy nabízí, a nabízí je vesměs všechny systémy, jsou uživatelské role, možnost exportu do různých typů souborů, rozdělení do projektů, sdílené překlady a hlavně napojení na různé překladače a kontroly pravopisu.

Nové systémy, které jsou oblíbené mezi uživateli, jsou často vytvářeny tak, aby byly pro uživatele co nejlépe přístupné, proto jsou často vytvářeny jako webové aplikace, které kladou důraz na responsivitu a funkčnost v různých webových prohlížečích. Díky tomu není pro uživatele složité se ke svým překladům dostat odkudkoli, a není tak nucen instalovat různé aplikace na různé platformy. Jednoduše otevře svůj oblíbený internetový prohlížeč a vidí vše, co potřebuje.

Většina systémů nabízí výhody jako jsou uživatelské role, díky kterým každý uživatel dělá jen sovu dílčí část práce a překlady jsou díky tomu rychleji, kvalitněji a hlavně lépe přeložené. Další výhodou těchto systémů jsou automatické a sdílené překlady, díky kterým má uživatel možnost použít překlad, který už v databázi existuje, nebo použít překlad, který mu je nabídnut pomocí strojového překladu. Díky tomu jsou tyto překlady snadněji přeloženy, ale také zde nedochází k překlepům a chybám v pravopisu. Což je další výhoda, kterou tyto systémy disponují, kontrola pravopisu, pomocí které je uživatel kontrolován, zda nenapsal nic pravopisně špatně. Kontrola pravopisu je často doplněna v případě chyby i o návrhy správného textu, podobně jako je tomu u známých textových editorů.

Dále v této kapitole bude představeno několik různých řešení zabývající se správou překladů. U každého budou popsány jeho hlavní funkce, které uživatelům nabízí.

2.1 Lokalise

Lokalise je produkt a společnost, založená v roce 2016, která sídlí v Londýně. Společnost má přes 80 zaměstnanců a jejími nejvýznamnější klienty jsou například britská fin-techová společnost Revolut,

hudební streamovací služba Tidal nebo společnost Tesco.

Hlavním produktem je systém pro správu překladu, který umožňuje spravovat překlady pro mobilní aplikace, hry, software nebo digitální obsah v několika jazycích. Lokalise je vytvořený, jak sami uvádí, hlavně pro agilní týmy, které požadují automatizaci svých procesů překládání a lokalizace. Poskytují vývojářům nástroje, které umožňují snadnější práci při lokalizaci, jako třeba, výkonná API rozhraní, CLI nástroj, mobilní SDK a komplexní dokumentaci. Mezi další výhody, které systém nabízí je čisté rozhraní s možností automatického nahrání projektu s rozpoznáváním textu.

2.1.1 Společný překlad

Jedna z hlavních funkcí, kterou Lokalise nabízí je funkce *Společný překlad*, která umožňuje nepřetržitě pracovat v týmech na překladech digitálního obsahu. Díky různým uživatelským rolím, nastavené manažery, není tolik práce na samotných vývojářích. Proces překladu je tak strukturován na několik uživatelů, kde každý přiloží svou ruku k dílu. Systém nabízí několik úrovní přístupů i role v týmech nebo projektech, jako například správci, fakturanti, členové a vlastníci. Překladatelům je nabídnuta řada funkcí jako je databáze překladů, strojový překlad a nebo kontroly kvality a správnosti.

Uživatelé se dají seskupovat do různých logických skupin, kterým se dají nastavovat různé úrovně přístupů jak k jazykům tak funkcím. Skupiny uživatelů lze snadno přidávat k úkolům nebo projektům, následná jejich editace je také velice jednoduchá.

Funkce společný překlad rovněž umožňuje uživatelům komunikovat mezi sebou, pomocí integrovaného komunikačního nástroje.

2.1.2 Přehled a plánování práce

Jedná se o nástroje pro správu lokalizačních pracovních toků, který umožňuje týmům produktivně, flexibilně a efektivně spolupracovat, spravovat a kontrolovat překlady. Eliminuje zbytečné čekání pomocí zřetězených úkolů, kdy jsou tyto úkoly uspořádány za sebou, takže po jejich dokončení se automaticky spustí další úkol v řadě.

Tato funkce dovoluje podobně jako například Git verzovat obsah. Dovoluje je slučovat, porovnávat a nabízí automatické identifikování konfliktů, které mohou při verzování vznikat.

Nabízí přehled dat o daném projektu, na řídícím panelu projektu lze vidět stav dokončení, zbývající slova k překladu, neověřené problémy a další. Nabízí přístup ke statistikám, kde si uživatel může zobrazit statistiky každého projektu v reálném čase. Může vidět slova přeložená do jednotlivých jazyků, členy týmu na projektu, počet klíčů, procento dokončených překladů a další. Také umožňuje uživateli vytvářet vlastní stavy, pomocí kterých popíše stav jakéhokoli překladu. Může si zvolit vlastní slova a barevně kóduje stav, což usnadní přehled.

2.1.3 Automatický proces lokalizace

Lokalise nabízí pokročilé nástroje pro automatizaci lokalizace, které umožňují bezproblémovou integraci lokalizace do pracovního toku automatického nahrávání tak, aby uživatel nemusel manuálně nahrávat nebo stahovat aktuální překlady. Sytém nabízí možnost nastavit vlastní pravidla, pro automatické aktualizování překladů projektu, systém tak automaticky provede určité akce na základě identifikovaných změn.

Systém nabízí integraci úložiště kódu s možností automatického načtení, takže Lokalise umí okamžitě importovat aktualizace jazykových souborů přímo do zvoleného repozitáře, nabízí integraci se systémy jako je Bitbucket, Github nebo Gitlab. Pomocí rozhraní API může vytvářet řetězce jakékoli složitosti. Exportuje sdružené překlady nebo jednoduše seznam překladů jako JSON. Umožňuje také použít nástroj CLI, který lze integrovat přímo do skriptů CI / CD, Jenkins nebo ho jednoduše použít jako nástroj pro nahrávání. Nabízí možnost uložit jazykové překlady přímo na cloudové úložiště, díky tomu lze nastavit přímé nahrávání do Amazon S3 nebo Google Cloud Storage.

2.1.4 Zabezpečení kvality překladu

Nástroje nabízené společností Lokalise pro zajištění kvality překladů umožňují zajistit, aby nedocházelo k chybám v překladech a zároveň usnadňují uživatelům jejich práci. Díky automatizovaným, konfigurovatelným kontrolám kvality překladu, lze snížit počet chyb. Pomocí automatické paměti překladů, lze ušetřit čas a zajistit vzájemnou konzistenci v překladech, a to pomocí shody v použitých překladech. Lze využít i automatického překladu, díky využití nejlepších nástrojů pro automatický strojový překlad od společností Google, DeepL nebo Microsoft. Integrované kontroly pravopisu a gramatiky jsou nabízeny pro dvacet dva světových jazyků.

Pro zajištění jednoznačnosti pojmů, systém nabízí možnost popsání a definování každého termínu ve vícejazyčném glosáři. Tím lze zajistit konzistentnost překladů projektu.

2.1.5 Cena

Lokalise nabízí tři základní cenové balíčky, které si uživatel může zvolit. K tomu nabízí další balíček *Enterprise*, který je určen hlavně pro velké společnosti. První základní, nejlevnější balíček, se jmenuje *Start* a stojí $ 90 měsíčně. Je směřován hlavně na malé týmy a začínající startupy. Nabízí neomezené množství projektů, webový editor pro překlady, tasky, možnost připojení na API, integraci do GitHubu, GitLabu nebo BitBucketu, dále integraci do Slacku, Jiri a podobných komunikačních kanálů a hlavně umožňuje uživatelům uložit a pracovat s 5000 klíči. Druhý, prostřední balíček, je nazván *Essentila* A stojí $ 190 měsíčně. Uživatelům nabízí stejné výhody jako balíček start a k tomu navíc funkce jako glosář, screenshoty, paměť překladů, historii překladů, strojový překlad, integraci do WordPressu, statistiky a reporty nebo aktivitu uživatelů na projektu. Proti balíčku start nabízí využití až 10 000 různých klíčů pro překlad. Posledním ze základních tří balíčku, je balíček *Pro*, který je nabízen za $ 435 za měsíc. Ten nabízí to stejné jako nižší balíček Essential a k tomu navíc

funkce jako dělení do větví, plugin pro Adobe XD, Sketch nebo Figmu, integraci do Google Cloud Storage a Amazon S3, vlastní stavy překladů, uživatelské skupiny, sdílené glosáře nebo asistenci při prvním nastavení. Nabízí až 30 000 různých klíčů s překlady. Balíček pro firmy je dost individuální, výhodou je vlastní vyhrazený manažer, konzultace a popřípadě vlastní funkce podle požadavků. [1]

2.2 Phrase

Phrase je platforma pro lokalizaci softwaru, zejména pro webové stránky nebo webové aplikace. Společnost byla založeno v roce 2012 v Hamburku a jeho tým tvoří lidé z celého světa.V současné době jejich produkt využívá tisíce podniků z více než 60 zemí světa. Mezi nejvýznamnější klienty patří společnost uShip, která provozuje online tržiště přepravních služeb, dále francouzská společnost The Conversation nebo startup Kreditech.

2.2.1 Integrace a přizpůsobení

Phrase dovoluje svým uživatelům využít rozsáhlé API, díky kterému je možné importovat a stahovat lokalizační soubory. To umožňuje developerům interagovat různými způsoby přímo s daty uloženými v systému. Další možnost integrace je pomocí CLI, kdy systém dovoluje rychlý přístup k projektům a překladům pomocí příkazové řádky. Možná je i synchronizace s GitHub, GitLab nebo Bitbucketem. Designérům, kteří spolupracují na projektu je umožněna synchronizace přímo s grafickým návrhovým softwarem Sketch, nebo Figma, díky tomu lze lokalizaci přidat přímo do návrhu aplikace. Integrace pro emailové upozornění nebo pro upozornění pomocí aplikací třetích stran, jako třeba Slack nebo Jira, umožňují uživatelům zůstat informováni o různých akcích v rámci projektu.

Systém samozřejmě nabízí uživatelské role a řízení uživatelského přístupu. Uživatelské role definují oprávnění pro různé skupiny uživatelů. Phrase nabízí několik uživatelských rolí, které definují práva uživatelů. Ta jsou definována pro každého uživatele. V rámci různých projektů může mít uživatel jiné oprávnění a jinou roli. Celkem v systému existuje pět různých rolí plus jedna role všemocného vlastníka, který má přístup do celého systému.

Phrase nabízí možnost exportovat a importovat více než 40 různých typů souborů. Navíc díky systému pro ověřování formátu souboru, který ověřuje soubor při samotném nahrávání, eliminuje nefunkční nebo nepodporované soubory. Umožňuje také bezdrátové aktualizace, při kterých aplikace v reálném čase a bez čekání, je schopna publikovat novou verzi překladů.

2.2.2 Zlepšování kvality překladu

Pomocí uživatelsky přívětivému editoru pro překlady a jejich klíče, je jednoduché získat pro překladatele data, které mu dopomáhají k překladu. Systém nabízí totiž možnost přeložit klíče v maximálním kontextu a to díky intertextovému editoru pro weby, pomocí kterému lze vidět v reálném

čase jak překlady vypadají přímo na webu. Druhá možnost pro přeložení do správného kontextu je díky screenshotů, které mohou být u překladu doplněny. Díky těm může překladatel vidět jakou část aplikace zrovna překládá a z ostatních překladů vyvodit její kontext.

Systém samozřejmě pracuje s automatickým strojovým překladem, díky kterému uživateli urychlí proces překladu.Uživatel má také možnost využít chytrého návrhu. Pomocí automatického vyplňování, který systém nabízí, usnadní uživatelům práci. Ti pak mohou pomocí systému kontrol, kdy systém nové překlady, nebo překlady u kterých byl změněn zdrojový obsah, automaticky označí jako neověřené a uživatel pak tyto neověřené překlady může vyfiltrovat, projít a zkontrolovat.

Uživatel může v editoru využít glosář, kde mohou být definovány výrazy, které jsou jedinečné pro firmu nebo značku, tak aby v rámci všech překladů zůstali konzistentní ve všech jazycích a projektech. Dále si může zobrazit historii všech předchozích verzí překladu a případně podle potřeby použít předchozí překlad.

2.2.3 Zvyšování efektivity

Systém nabízí možnost organizovat práci do tasku a ty přiřazovat různým členům týmu. Díky čemuž lze dosáhnout zlepšení efektivity v překládání. Každému tasku lze přiřadit termín a jazyk, který má být v rámci úkolu splněn. Systém nabízí komentáře a označení, pomocí kterých lze komunikovat přímo v systému bez nutnosti používat jiný komunikační kanál.

Systém umožňuje monitorovat aktivitu, kterou lze filtrovat podle uživatelů, událostí anebo časových intervalů a díky tomu může být uživatel neustále informován o výkonech členů týmu včetně informací o množství přeložených slov, klíčů a dalšího.

K překladům lze také přidávat tagy, díky kterém může být projekt přehlednější a dá se lépe sledovat. Systém také nabízí verzování a práci ve větvích podobně jako v Gitu.

2.2.4 Cena

Pharse nabízí dva základní balíčky plus jeden určený velkým společnostem. První základní balíček je balíček *Basic*, a stojí $ 23 měsíčně. Společnosti nabídne maximálně 5 uživatelů, 100 API dotazů za 5 minut a uživatele, jeden projekt, neomezené jazyk, uživatelské role a nastavení přístupů, jednoduché vyhledávání, řazení překladů, 40+ podporovaných souborových formátů. Druhý základní balíček *Advanced* stojí $ 35 měsíčně. Společnosti nabízí neomezené množství projektů, 1 000 API dotazů za 5 minut a uživatele, neomezené množství projektů, minimálně 5 uživatelů, pokročilé vyhledávání, automatické doplňování, glosář, historii, screenshoty, překladovou paměť, integraci do Sketch, Figmy, na GitHub, GitLab, BitBucket nebo WordPress, Jira a Slack. Balíček postavený na míru společnosti, nabízí to, co nabízí balíček Advanced a také vlastní podporu, správu účtů, sledování aktivity, vlastní uživatelské role, podnikové a vlastní integrace, reporty a podobné funkce, šité na míru společnosti. [2]

2.3 Localazy

Localazy je český systém pro správu překladu, jejíž majitelé mají sídlo v Brně. Jedná se o celkem mladou společnost, která byla založena na začátku roku 2020. I když se jedná o poměrně mladou společnost, má stále co nabídnout. Nabízí totiž uživatelům funkce jako uživatelské role, automatické návrhy překladů nebo třeba integraci přímo do uživatelského projektu, který může být vytvořený například jako Android aplikace, React aplikace, Vue a mnohé další.

2.3.1 Uživatelské role

Localazy podporuje různé uživatelské role pro uživatele v rámci projektů. Díky tomu dovoluje uživatelům efektivně spolupracovat a zlepšit tak i kvalitu požadovanou pro překlady. V rámci projektu existuje šest různých uživatelských rolí. Role, která má nejvyšší práva, je nazvaná **majitel**. Ten může dělat vše, co je v projektu k dispozici - spravovat integrace, fakturaci, přispěvatele a nastavení projektu. Majitelé mohou také bez souhlasu překládat nebo kontrolovat. Další role je role **správce**, ten může upravovat role jiných uživatelů a pozvat uživatele do projektu, jinak má stejná oprávnění jako **recenzent**, což je další uživatelská role v systému. Recenzent má povolenou funkci kontroly, jeho úkol v projektu je hlavně kontrolovat a schvalovat vytvořené překlady. Také může upravovat glosář. Další dvě role jsou **překladatel** a **důvěryhodný překladatel**. Ti vytvářejí překlady. Rozdíl mezi nimi je, že důvěryhodný překladatel nepotřebuje kontrolu nad svými překlady. Poslední role je **žádný vztah**, která slouží pro odebrání uživatelů z projektu.

2.3.2 Editace překladů

Rozhraní formuláře se umí přizpůsobí typu fráze, kterou uživatel upravuje (řetězec, pole nebo množné číslo). V rámci editoru může uživatel vidět souhrn kontextu frází, které obsahuje aktuální překlad, poznámku a stav fráze. Dále pak metadata jako je cesta nebo soubor, kde se fráze nachází. Kterýkoli správce může tyto hodnoty upravovat tak, aby pro překladatele byly co nejaktuálnější a obsahovali co nejpřesnější data. Překlad množných čísel je jednoduchý, protože dokáže editor přizpůsobit tomuto úkolu. Navíc Localazy podporuje několik jazykových verzí množných čísel, čímž překladatelům značně usnadňuje práci.

Pro lepší překládání a zachování významu ve všech jazycích, přichází Localazy s funkcí, která vyžaduje kontrolu ve všech použitých jazycích. To znamená, že uživatel změní překlad, označí ho za změněný a všechny ostatní jazykové verze budou nyní vyžadovat změnu, nebo kontrolu.

2.3.3 Glosář

Glosář je základním nástrojem pro zajištění vysoce kvalitního překladu a poskytnutí kontextu překladatelům. Díky termínům definovaném v glosáři, mohou překladatelé zajistit jednotné přeložení daných frází taky, aby zůstal zachován význam a bylo zajištěno to, zda se má daný výraz překládat

či nikoli, nebo jak mají být psané malá a velká písmena. To vše lze v glosáři zaznamenat pro fráze, u kterých to uživatel vyžaduje.

2.3.4 Sdílení překladů

Localazy nabízí možnost sdílení překladů tak, aby si uživatelé mohli navzájem pomoci při překládání. Když je nějaký překlad schválen, sdílí se a distribuuje mezi všechny ostatní aplikace na Localazy. Uživatel si pak může vybrat z několika různých verzí pečlivě připravených překladů. Může také snadno aktualizovat text podle svých potřeb a vytvořit zcela nový překlad ze stávajícího. Díky tomu může uživatel rychleji a přesněji překládat do nových jazykových verzí.

2.3.5 Chytrá paměť překladů

Chytrá paměť překladů skenuje aplikaci a hledá stejné fráze ve zdrojovém jazyce s chybějícími překlady v jiných jazycích. Pokud chybí překlad, který by mohl být dokončen z jiného zdroje, je nabízen prostřednictvím kontroly. Lze se tak vyhnout překládání stále stejných frází dokola.

2.3.6 Cena

Lokalazy oproti ostatním systémům má ceník nastavený jinak. Platí se zde za počet uložených frází. Navíc tento poplatek je jednorázový a fráze se dají postupně přikupovat. Prvních 200 frází je zdarma, poté se dá přikoupit 100 frázi za $ 15, 250 za $ 30, 500 za $ 50, 1 000 za $ 90, 5 000 za $ 400 a nebo 10 000 za $ 750. [3]

Kapitola 3

Přehled moderních technologií pro tvorbu webových aplikací

Tato kapitola bude zaměřena na moderní technologie pro tvorbu webových aplikací. Na úvod je důležité si ujasnit, co to webová aplikace je. Jedná se o aplikaci, kterou není nutné instalovat na zařízení uživatele, uživatel ji jednoduše spustí pomocí svého internetového prohlížeče, bez nutnosti cokoli instalovat. Na první pohled se jedná o webovou stránku, která je tvořena složitějším, dynamicky měnícím se obsahem. V dnešní době jsou webové aplikace často schopny nahradit desktopové aplikace hlavně díky rychlému internetu a dostatečnému výpočetnímu výkonu zařízení, na kterých aplikace běží.

Požadavky na webové aplikace stále rostou, a proto tomu rostou i požadavky na technologie jako takové. Základní technologií při tvorbě webové aplikace je již po desetiletí HTML a CSS, díky které se vytváří vzhled aplikace samotné. HTML je značkovací jazyk, který slouží k tvorbě webových stránek a obsahuje širokou škálu značek (tagů), kde každá značka dále obsahuje několik atributů, na základě kterých se vykreslí stránka jako taková. CSS je způsob zápisu stylu zobrazovaných elementů na webové stránce tvořené HTML.

Pár let zpátky bylo pro vytváření webové aplikace hojně využíváno PHP, později s použitím známého frameworku Laravel. PHP je skriptovací jazyk, který je určen pro programování dynamických webových stránek. Skripty PHP jsou spouštěny na straně serveru a uživateli je posíláno samotné dynamicky vytvořené HTML. Laravel je, jak už je výše zmíněno, PHP framework sloužící pro webové aplikace, postavený na architektuře MVC. Jedná se o architekturu, rozdělenou do tří nezávislých komponent - aplikační, uživatelské a datové. V případě Laravelu Model obsahuje funkce a data, view slouží k zobrazení dat uživateli a Controller obstarává spojení mezi uživatelem, modelem a view. [4]

Při tvorbě dnešních moderních webů je nejčastěji používaný javascript nebo různé frameworky, které javascript využívají. JavaScript je skriptovací jazyk, který se dá využít multiplatformně, ať už při vytváření webové aplikace, nebo na serveru, kde je vhodný pro tvorbu serverových aplikací,

které pracují v reálném čase. Pro takové implementace se rovněž využívají různé frameworky jako například Node.js. Na frontendové části aplikace se často setkáme s frameworky nebo knihovnami jako ReactJS, Vue.js a nebo třeba AngularJS. A na tyto frameworky a knihovny bude zaměřena tato kapitole.

3.1 JavaScript frameworky a knihovny

JavaScript jako objektově orientovaný programovací jazyk, který slouží pro provádění výpočtů a manipulaci s výpočetními objekty v hostitelském prostředí. Byl původně navržen jako webový skriptovací jazyk, který poskytuje mechanismus k oživení webových stránek v prohlížečích a k provádění výpočtu serveru jako součásti webové architektury klient-server. Využití JavaScriptu přesáhlo rámec jednoduchého skriptování a nyní se používá pro celé spektrum programovacích úkolů v mnoha různých prostředích. Při rozšíření JavaScriptu se rozšířily také funkce a vybavení, které poskytuje nyní, proto se stal plně vybaveným univerzálním programovacím jazykem.

U webových aplikací prohlížeč poskytuje JavaScriptu prostředí pro výpočet na straně klienta, včetně objektů, které představují například okna, textové oblasti, rámečky, soubory cookie a podobné objekty. Hostitelské prostředí je v nejčastějším případě webový prohlížeč poskytující také prostředky pro připojení kódu k událostem, jako je načítání stránek nebo obrázků, změna velikosti obrazovky, chyby, odeslání formulářů nebo akce myši. Výsledná aplikace poté obsahuje HTML, ve kterém je vložený skriptovací kód a zobrazená stránka je kombinací prvků pevného uživatelského rozhraní a dopočítaných elementů. Skriptovací kód reaguje na interakci s uživatelem a není potřeba žádný hlavní program. [5]

Framework Javascriptu definuje celý design aplikace. Některé frameworky JavaScriptu jsou postaveny na architektuře MVC určené k oddělení webové aplikace do nezávislých jednotek tak, aby se zlepšila kvalita a udržitelnost kódu. Mezitím co javascriptová knihovna nabízí funkce, které mají být volány jejím nadřazeným kódem, framework definuje celý design aplikace. Vývojář nevolá framwork, místo toho framwork volá a používá kód nějakým konkrétním způsobem. Knihovna JavaScriptu umožňuje snadnější vývoj aplikací, zejména pro AJAX a další technologie zaměřené na web. Některé knihovny JavaScriptu umožňují snadnější integraci s jinými technologiemi pro vývoj webových aplikací jako jsou CSS, PHP, Ruby a Java. Téměř všechny knihovny JavaScriptu jsou vydávány na základě licence copycenter nebo copyleft tak, aby byla zajištěna bezlicenční distribuce, použití a úpravy.

3.2 ReactJS

React je open-source, front-endová knihovna JavaScriptu pro vytváření uživatelských rozhraní nebo komponent uživatelského rozhraní. Je vyvíjen Facebookem a komunitou jednotlivých vývojářů. Základ celého Reactu jsou komponenty, které představují zapouzdřené celky s danou funkcionalitou,

které lze znovu přepoužít. Tyto komponenty jsou dále používány jako samostatné HTML elementy s danými vlastnostmi a vlastním vnitřním stavem. React je specificky zaměřenou knihovnou, zabývá se pouze správou stavu a vykreslením tohoto stavu do DOM, takže vytváření aplikací React obvykle vyžaduje další knihovny, například pro směrování. Příkladem takové knihovny je React Router, která udržuje UI prostředí aplikace synchronizované s URL adresou a řeší routování mezi různými stránkami aplikace a dále manipuluje a pracuje s adresou URL.

3.2.1 JSX

Jádrem každé základní webové stránky jsou dokumenty HTML. Tyto dokumenty tvoří v prohlížeči DOM reprezentující to jak daná stránka vypadá. V Reactu je však klasické HTML nahrazeno JSX. Díky tomuto rozšíření je možné použití k aktualizaci DOM funkci zvanou Virtual DOM. Toto vede k významnému zlepšení výkonu webu a efektivnějšímu vývoji.

3.2.2 Virtual DOM

React JS vytváří něco, čemu se říká virtuální DOM. Virtuální DOM je kopií DOM daného webu a React JS pomocí této kopie zjistí, jaké části skutečného DOM je zapotřebí změnit, když dojde k události, která tuto změnu vyžaduje. Při takové události React prohledá virtuální DOM a selektivně aktualizuje pouze tu část, která tyto změny vyžaduje. Tento druh selektivní aktualizace používá menší výpočetní výkon a trvá kratší dobu načítání.

3.2.3 React komponenta

Na začátek je nutno říct, že React v tuto chvíli využívá dva způsoby pro tvorbu komponent. První způsob, poněkud starší, a u nových projektů se již většinou nepoužívá, je zápis komponent pomocí tříd. Druhý, novější způsob, je zápis komponent do funkcí, kde dochází k využívání tzv. hooks.

```
class Date extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
    this.setState({ date: this.props.date });
  }

  render() {
   const { date } = this.state;
```

```
    return (
      <div>
        <span>Date: {date}</span>
      </div>
    );
  }
}
```

Listing 3.1: Ukázka komponenty zapsána pomocí třídy

V ukázce lze vidět příklad komponenty zapsané pomocí Class zápisu. Komponenty vytvořené tímto způsobem poskytují několik metod, které vznikají rozšířením třídy *React.Component*. Metody převzaty z této třídy mohou být přepsány a díky tomu umožňují spustit kód a různé funkce v konkretních časech procesu, takzvané životní cykly. Životní cykly komponenty jsou tři. První je její načtení (mounting), při které jsou volané metody jako *constructor()*, *componentDidMount()*. Další životní cyklus komponenty je aktualizace (updating). Tento cyklus nastane při změně parametrů předávaných do komponenty nebo při změně vnitřního stavu, který musí byt upraven metodou *setState*. Při aktualizaci je volána metoda *componentDidUpdate()*. Posledním životním cyklem komponenty je její odstranění (unmounting), které nastává při odstranění komponenty z DOM je při ní volána metoda *componentWillUnmount()*.

```
function Date({ initDate }) {
  const [date, setDate] = useState(new Date());

  useEffect(() => {
    setDate(initDate);
  }, []);

  return (
    <div>
      <span>Date: {date}</span>
    </div>
  );
}
```

Listing 3.2: Ukázka komponenty zapsána pomocí funkce

U komponent vytvořené pomocí funkce jsou často využívány takzvané *React Hooks*, které umožňují reagovat na stav a životní cyklus komponenty. Dva základní hooky, které React využívá jsou *useState* a *useEffect*. Hook *useState* je používán pro uchování hodnoty a díky němu komponenta reaguje na změnu stavu a tak může být znovu vykreslena. Jedná se o obdobu *setState* u třídního

zápisu. Na rozdíl však od *setState useState* nemusí obsahovat celý *state* (objekt), ale může obsahovat jednu hodnotu a v komponentě může být použito více *useState*. *UseEffect* je hook, který slouží ke stejnému účelu jako *componentDidMount*, *componentDidUpdate* a *componenWillUnmount*, záleží na jeho zápisu. Má dva argumenty. První je pro funkci, která je volána na základě změny stavu. Druhém argumentu tvoří pole závislostí. Pole závislostí však není povinné a v takovém případě je *useEffect* volán pokaždé. Pokud je parametr závislostí tvořen prázdným polem, *useEffect* slouží jako *componentDidMount*. Return funkce v takovém případě je jako *componentWillUnmount*. Když je pole závislostí naplněno proměnnými, slouží takový *useEffect* jako *componentDidUpdate*. React umožňuje tvořit vlastní hooky, které jsou hojně využívány dalšími používanými knihovnami. [6]

3.3 Vue.js

Vue je progresivní framework pro tvorbu uživatelského rozhraní. Na rozdíl od jiných frameworků je Vue navržený tak, aby se dal znovu postupně použít. Základní knihovna je totiž zaměřena pouze na zobrazovací vrstvu a díky tomu je jednoduché ji integrovat s dalšími knihovnami nebo stávajícími projekty. Vue je ale také schopné vytvářet a držet při chodu sofistikované single-page aplikace, je-li používáno s moderními nástroji a dalšími pomocnými knihovnami. Tyto knihovny často u aplikací řeší směřování, správu stavů a vytváření nástrojů.

3.3.1 Vue zápis

Vue je tvořen klasickým HTML zápisem, do kterého jsou pomocí deklarativního vykreslování vkládána data nebo funkce, které jsou definovány samotnou Vue instancí. Vue kompiluje tyto HTML šablony do svého virtuálního DOM. Díky virtuálnímu DOM je podobně jako React schopný renderovat jen ty komponenty, které to vyžadují. HTML elementy jsou doplněny o řadu dalších atributů, které jsou označený předponou "v-". Tyto atributy slouží k dynamickému chování a udržení aktuálních dat v komponentě.

3.3.2 Data a jejich správa

Vue drží svá data v každé své instanci (komponentě) zvlášť. V rámci komponenty se přistupuje k datům pouze přes název parametru. V logické části komponenty je přistupováno k datům v rámci objektu přes proměnnou *this*. Data lze samozřejmě vypsat přímo do šablony komponenty. Výhodou je že výpis se automaticky escapuje, proto nemusí být ošetřen a nehrozí, že by zobrazena data v šabloně obsahovala nebezpečný HTML kód nebo script. Při potřebě vypsat data bez escapování se dá využít atributu "v-html". K atributům vlastní komponenty (props) se přistupuje stejně jako k datům. K předávání dat do klasického HTML atributu slouží dvojtečková anotace, díky ní Vue pozná, že se jedná o proměnou nebo parametr a ne řetězec.

Často je při tvorbě potřeba vypsat data v požadovaném formátu, nebo je potřeba vytvořit proměnnou tvořenou více proměnnými. K tomu nám ve Vue slouží tzv. *Computed*, ten je schopný pomocí jedné proměnné vypsat data, která jsou naformátovaná nebo složená z více proměnných. Vypočítává se vždy při vykreslení a proto jsou data pokaždé aktuální. Další často využívanou metodou, je metoda *Watch* umožňující elegantní reakci na změnu dat. V případě změny proměnné je schopna zavolat další funkce nebo metody, které jsou definované uživatelem.

3.3.3 Podmínky a cykly

Vue přináší jednoduchý způsob, jak podmínky vložit přímo do části šablon, a to přímo jako atribut HTML elementů. To znamená, že přidáním atributu "v-if"do jakékoli komponenty, která kontroluje hodnotu vloženou do atributu, se na základě toho zobrazí nebo nezobrazí element a všechny jeho podřazené prvky. V momentě potřeby vytvořit if - else, se používá stejný zápis jako v předchozím případě. Je důležité, aby element s atributem "v-else"byl na stejné úrovni jako element s atributem "v-if".

Pro procházení dat obsahuje Vue jednoduchý a efektivní způsob jak procházet pole dat. Pomocí atributu "v-for"v elementu je schopen vytvořit seznam dat, která jsou uložená v poli.

```
<ul>
   <li v-for="item in array">
      {{ item }}
   </li>
</ul>
```

Listing 3.3: Ukázka cyklu zapsána ve Vue

3.3.4 Vue komponenta

Vue komponenta je v podstatě instance Vue s předdefinovanými vlastnostmi, kterou stačí jednoduše zaregistrovat a dá se ihned použít. Každá komponenta se dá použít na několika místech a v době vykreslení každá zvlášť vytvoří novou, oddělenou instanci, která drží svůj vlastní stav. Díky tomu se každá chová nezávisle. Komponenta si nese vlastí HTML šablonu, ve které může vykreslovat data. Vlastní aplikační logiku a data, komponenty mohou být použity všude a jakýmkoli způsobem a jednoduše přijímají a zobrazují data nezávisle na sobě.

Každá komponenta je schopna detekovat a zpracovávat události vyvolané uživatelem. Ať už jde o kliknutí na tlačítko, doplnění textu nebo přejetí přes nějaký prvek. Princip zachycování událostí je založený na posluchači událostí nad HTML elementy, který je následně rozšířen pomocí Vue. Události ve Vue lze zachytávat dvěmi způsoby, buď přes *v-on:názevUdálosti* (např. *v-on:click*) nebo pomocí zkráceného zápisu *@názevUdálosti* (např. *@click*). Uvnitř události lze volat přímo potřebnou metodu, nebo zapsat jednoduchý kód a provést akci přímo. V takovém případě nemusí být v kódu pro

přístup k proměnným použita proměnná *this*. Pro přístup k metodám se však musí použít proměnná *this*. Vue však nabízí velmi zajímavé řešení přímo pro formuláře, kde každé pole ve formuláři musí mít svou hodnotu reprezentovanou proměnnou. Pomocí konstrukce *v-model* se data přesouvají tam a zpět, a díky tomu dochází k jednodušší manipulaci dat a dá se takto elegantně řešit například dynamická validace dat.

Podobně jako React i Vue komponenta má své životní cykly pro práci s daty. Ve Vue jsou tyto cykly nazvány *beforeCreate*. Je volán bezprostředně po inicializaci instance ještě před vykreslením dat a nastavení událostí posluchačů. Další je textitcreated, který je volán po vytvoření instance a nastavení posluchačů události. *BeforeMount* se volá těsně před zahájením renderování komponenty a poslední metoda, která je volána na začátku životního cyklu komponenty je metoda *mounted*, která je volána poté, co se komponenta vyrenderuje. Další fáze, při které jsou volány metody spojené s životními cykly, je update. Při updatu jsou volány metody *beforeUpdate* a *updated*. Při odstranění komponenty z DOM jsou volány metody *beforeUnmount* a *unmounted*.[7]

3.4 Angular

Jedná se o open-source webový aplikační framework vedený a podporovaný společností Google, která tento framework založila v roce 2009. Aplikace vytvořeny v Angularu jsou tvořeny pomocí HTML kódu, do kterého jsou podobně jako u Vue bindovány (vkládány) pomocí speciálního zápisu/značek operace nebo data.

Jedná se o druhou verzi frameworku. První verze byla postavena na JavaScriptu, přičemž druhá verze již vychází z TypeScriptu. Není to však jediný rozdíl mezi AngularJS (první verzí) a Angularem (označovaný také jako Angular 2+), kterou Google navrhl při přepisu na novou verzi. Angular například neobsahuje koncept "scope"nebo ovladačů. Místo toho používá hierarchii komponent jako primární architekturu, rovněž používá pozměněný zápis pro vlastnosti, kde používá "[]"nebo pro vazbu událostí, kde zase využívá "()". Angular také doporučuje použit jazyk TypeScript od společnosti Microsoft, který přináší spoustu dalších funkcí jako statické typování včetně obecných typů nebo anotace, TypeScript je nadmnožinou ES6, která je zpětně kompatibilní s ES5.

3.4.1 TypeScript

Angular je framework postavený na jazyku zvaný Typescript. Jedná se o open-source jazyk vytvořený a spravovaný společností Microsoft. Je postavený na JavaScriptu, do kterého přidává definici statických typů. Ty poskytují způsob, jak popsat data v objektu, poskytnout lepší dokumentaci a umožňují ověřit, že kód funguje správně. Samotný kód TypeScriptu se kompiluje do JavaScriptu.

Hlavními vlastnostmi TypeScriptu jsou anotace typů a typová kontrola, definování typů u rozhraní, tříd a modulů, výčtové typy (*enum*) nebo výchozí hodnoty parametrů funkcí.

3.4.2 Angular komponenta

Komponenty jsou základní stavební bloky, které tvoří aplikaci. Komponenta zahrnuje třídu TypeScript s dekorátorem *@Component()*, šablonou HTML a styly. Dekorátor *@Component()* určuje CSS selektor, který definuje, jak komponenta používá v šabloně prvky HTML. Ty které v šabloně odpovídají tomuto CSS selektoru, se stanou instancemi komponenty. Deklarátor dále obsahuje šablonu HTML, která instruuje Angular, jak vykreslit komponentu a volitelnou sada stylů CSS, které definují vzhled prvků HTML šablony.

3.4.3 Šablona komponenty

Každá komponenta má šablonu HTML, která deklaruje, jak se tato součást vykresluje. Tuto šablonu definujete přímo nebo jako cestu k souboru. Angular rozšiřuje HTML o další syntaxi, která umožňuje vložit dynamické hodnoty z komponenty. Angular automaticky aktualizuje vykreslený DOM, když se změní stav komponenty.

Pro začlenění dynamické hodnoty do řetězce se dá v Angular šabloně použít interpolace textu, díky ní se může dynamicky měnit vykreslovaná hodnota v aplikaci. Takový zápis pak vypadá následovně *{{ value }}*. Při potřebě v aplikaci zachytávat různé druhy událostí, vyvolané uživatelem na některém z elementů komponenty, se používá tento zápis *(event)="metoda()"*, který umožňuje na jakýkoli event zavolat vlastní vytvořenou metodu. Pro dynamické nastavení vlastností elementu se v Angularu používá tento zápis *[vlastnost]="hodnota"*. Dá se použít i zápis pro vkládání dynamických hodnot do textu (interpolace). V takovém případě by zápis vypadal následovně *vlastnost={{hodnota}}*.

Podobně jako ve Vue, umí Angular pracovat obousměrně s hodnotou ve formuláři pomocí konstrukce *NgModel*, jejíž zápis pak vypadá takto *[ngModel]="hodnota"*. Dále pak Angular umožňuje podmíněné renderování pomocí *NgIf*, když NgIf je false. Angular odebere prvek a jeho potomky z DOM. Angular poté zlikviduje své komponenty, což uvolní paměť a zdroje. Zápis pro použití této podmínky vypadá **ngIf="hodnota"*. Angular má i vlastní funkcionalitu pro vykreslování pomocí cyklu FOR, který Angular nazývá *NgFor* a používá se následovně **ngFor="let item of items"*. V rámci všech podřízených prvků pak má přístup k hodnotě uložené v proměnné *item*, která reprezentuje jeden záznam z pole *items*.

3.4.4 Životní cykly komponenty

Životní cykly fungují podobně jako životní cykly v Reactu či Vue. Instance komponenty má životní cyklus, který začíná, když Angular vytvoří instanci třídy komponenty a vykreslí pohled komponenty spolu s jeho podřízenými pohledy. Životní cyklus pokračuje detekcí změn, protože Angular kontroluje, zda se mění vlastnosti vázané na data, a podle potřeby aktualizuje pohled i instanci komponenty. Životní cyklus končí, když Angular zničí instanci komponenty a odebere její vykreslenou šablonu z DOM.

Angular reaguje na životní cykly pomocí Hooku, které jsou pojmenovány podle rozhraní a přidání předložky *ng* - například funkce onInit se nazývá *ngOnInit()*. Pokud je implementována tato metoda ve své třídě, Angular ji zavolá krátce po první kontrole vstupních vlastností pro tuto komponentu. Další důležitou metodou, je metoda nazvaná *ngOnChanges()*. Ta reaguje, když Angular nastaví nebo resetuje vstupní vlastnosti vázané na data. Metoda, která je volaná na konci životního cyklu komponenty je v Angularu nazvána *ngOnDestroy()*.

3.4.5 Dependency injection

Vkládání závislostí umožňuje deklarovat závislosti tříd TypeScript bez toho, aby se staraly o jejich instanci. Instalaci místo toho vytvoří Angular sám. Tento návrhový vzor umožňuje psát více testovatelný a flexibilní kód. [8]

3.5 Ember

Ember.js je MVC (Model-View-Controller) framework JavaScriptu používaný k vývoji velkých webových aplikací na straně klienta. Zahrnuje vše, co je potřeba k vytvoření bohatých uživatelských rozhraní, která fungují na jakémkoli zařízení. Dělá to tak, že poskytuje vývojářům mnoho funkcí, které jsou nezbytné pro správu složitosti moderních webových aplikací, a také integrovanou sadu nástrojů pro vývoj, která umožňuje rychlou iteraci.

Jelikož se jedná o MVC, je celá aplikace rozdělená do vrstev. Ember zde využívá další knihovny, které uživateli zjednodušují práci. Pro vrstvu Model, která je zodpovědná za veškerou komunikaci na straně serveru i za úlohy specifické pro model, jako je formátování dat, využívá "Ember data"k dramatickému zjednodušení kódu a zároveň ke zvýšení robustnosti a výkonu aplikace. Mezi modelem a pohledem je vrstva kontrolérů. Pro zjednodušení existuje několik řadičů, zejména "Ember.ObjectController"a "Ember.ArrayController". Ember Router je mechanismus, který aktualizuje adresy URL vaší aplikace a poslouchá změny adres URL. Každá trasa může mít několik dílčích tras a pomocí routeru může být přecházeno mezi stavy v aplikaci. Vrstva zobrazení je zodpovědná za kreslení prvků na obrazovce. Ember.js má propracovaný systém pro vytváření, správu a vykreslování hierarchie pohledů, které se připojují k DOM prohlížeče. Zobrazení jsou zodpovědná za reakce na události uživatelů, jako jsou kliknutí, tažení a posouvání, stejně jako aktualizace obsahu DOM, když se změní data, která jsou základem zobrazení.

3.5.1 Ember komponenta

Komponenta je opakovaně použitelný prvek uživatelského rozhraní skládající se ze šablony a volitelné třídy JavaScriptu, která definuje jeho chování. Například může být vytvořeno tlačítko v šabloně a zpracováno chování kliknutí v souboru JavaScript, který sdílí stejný název jako šablona. Komponenty jsou v Emberu rozděleny do dvou kategorií. První jsou komponenty bez JavaScriptu, které jsou

založeny pouze na šabloně. Druhé jsou komponenty s JavaScriptem, které se skládají ze šablony a třídy podpory.

I když je tohoto v Emberu dosaženo pomocí šablon HTML hodně, je potřeba tvořit komponenty s použitím JavaScriptu, aby byla aplikace více interaktivní. Pokud je potřeba tvořit dynamické hodnoty měnící se přímo v renderu, musí být použito *Tracked Properties*. Ty se tvoří takto *@tracked hodnota = 0;* a díky tomu se při každé změně aktualizuje šablona v DOMu s novou hodnotou. Výpis v šabloně pak vypadá takto * {{this.hodnota}} *. Hodnoty však často potřebují být měněny na základě různých událostí a k tomu slouží v Emberu modifikátor *on*. Modifikátory HTML jsou Emberova syntaxe, která umožňuje připojit k HTML tagu logiku. Vypadá takto *{{on "click"this.modifikator}}*. Ale aby tyto modifikátory událostí něco udělaly, budeme muset definovat akce v komponentě. Akce je metoda JavaScriptu, kterou lze použít ze šablony.

```
@action
modifikator() {
  this.hodnota = this.hodnota + 1;
}
```

Listing 3.4: Ukázka zápisu metody v Emberu

V šabloně komponenty může být použito podobně jako ve Vue či Angularu podmíněné vykreslení obsahu. Syntaxe vypadá následovně. Pokud je podmínka pravdivá, Ember vykreslí obsah, který je uvnitř bloku. Stejně jako mnoho programovacích jazyků Ember také umožňuje psát příkazy if else a if else if přímo v šabloně.

```
{{#if podminka1}}
  ...
{{else if podminka2}}
  ...
{{else}}
  ...
{{/if}}
```

Listing 3.5: Ukázka zápisu podmínky v Emberu

Pokud přichází potřeba komponentu opakovat několikrát za sebou a s různými daty pro každé použití komponenty, pak může být použito *{{#each}}* k procházení seznamů podobných položek a opakování části šablony pro každou položku v seznamu.[9]

Kapitola 4

Návrh a implementace systému

Celý systém byl vytvořen pomocí nejmodernějších nástrojů pro tvorbu moderní webové aplikace. Byly použity moderní knihovny, které pomáhají vývojářům usnadnit práci a uživateli ulehčují a zpříjemňují používání aplikace. Celý systém byl postaven na dvou aplikacích, a to na klientské aplikaci, která je spouštěna přímo ve webovém prohlížeči uživatele a komunikuje se serverem. Aplikace, která pak běží na straně serveru, se stará o zpracování uživatelských dat tak, aby byla bezpečně uložena a co nejrychleji k dispozici pro uživatele. Informace o použitých technologiích a knihovnách budou popsány v další podkapitole.

4.1 Použité technologie

Pro vývoj bylo využito třívrstvé architektury. Pro datovou vrstvu je využita databáze MySQL 8.0, která uchovává data uživatelů. Pro aplikační vrstvu je využitý ASP.NET Core, který zpracovává API dotazy z prezenční vrstvy, pro tuto je využita webová aplikace vytvořena pomocí JavaScriptové knihovny React.

4.1.1 Databáze

Pro uchování dat byla zvolena databáze MySQL 8.0. Jedná se o relační databázový model vytvořený švédskou firmou MySQL AB, která je nyní vlastněna firmou Oracle Corporation. Databázový server běží ve virtualizovaném kontejneru pomocí Dockeru. Jelikož zatím není databáze nijak rozsáhlá a nedochází zde k častým a obsáhlým dotazům na data, je toto řešení dostatečné. V případě rozšíření aplikace a nárůstu uživatelů by bylo nutné, aby databáze běžela na vlastním serveru. [10]

4.1.2 Serverová aplikace

Pro komunikaci mezi uživatelem a databází byla vytvořena aplikace pomocí ASP.NET, který je součástí .NET vyvíjen společností Microsoft. Serverová aplikace slouží hlavně ke zprostředkování dat

uživateli a k tomu využívá ke komunikaci s databází knihovnu *EntityFrameworkCore*. Ta umožňuje jednodušší přístup k datům, kde se přístup k datům provádí pomocí modelu. Model je tvořen třídami entit a objektem kontextu, který představuje relaci s databází. Kontextový objekt umožňuje dotazování a ukládání dat.

Dále serverová aplikace využívá knihovnu pro ověřování uživatelů pomocí JWT tokenu. Díky tomu může serverová aplikace určit, zda má uživatel, který se dotazuje na backend, právo na získání dotazovaných dat či nikoli.

Hlavním úkolem serverové aplikace je však odesílání a zpracovávání API dotazů z webové aplikace. Toto je řešeno pomocí třídy *ControllerBase*, která se stará o zpracování API dotazů, jež jsou následně zpracovány a data jsou poslány zpět do webové aplikace. Pomocí .NET je u každé metody určeno o jaký dotaz se jedná a zda jde o POST či GET. Lze také určit zda k získání dat musí být uživatel autorizovaný nebo ne. [11]

Swagger je framework pro návrh a tvorbu dokumentace, která na serveru umožňuje zobrazit a otestovat dotazy k tomu určeny. Proto pro každou metodu, která je v dokumentaci zapsána, je určen její návratový typ nebo to zda se má v dokumentaci objevit nebo ne. Díky této dokumentaci může být jednodušší pro další vývojáře se zorientovat v rozhraní pro API a tak může být toto rozhraní využito i třetími stranami.

4.1.3 Webová aplikace

Samotná webová aplikace byla vytvořena pomocí JavaScriptové knihovny React, která je podrobněji popsána v kapitole 3.2, společně s TypeScriptem. Ten se stará o statické typování atributů a proměnných a jejich typovou kontrolu.

Pro tvorbu uživatelského rozhraní byla využita knihovna *Material UI* od Material Design, která je vyvíjena a spravována společností Google. Tato knihovna obsahuje spoustu React komponent, která dávají možnost vytvořit přehledné uživatelské prostředí. Umožňuje nastavit vlastní barevné schéma aplikace, upravovat použité komponenty, tak aby splňovaly požadavky vývojářů a pomáhají při vytváření responsivní webové aplikace.

Webová aplikace využívá knihovnu *i18next*, která slouží pro lokalizaci celé aplikace. Tato knihovna využívá pro uchování překladů JSON soubory, kde každý překlad je uložen pod jedním klíčem. Tyto klíče jsou poté vkládány do funkce a generují texty odpovídajícím zvolenému jazyku a klíči.

Pro routování a práci s URL je využita knihovna *React router DOM*, která se stará o routování napříč aplikací, jelikož React samotný tvoří single-page aplikaci. Proto pro vytváření komplexnějších aplikací je často využívána tato knihovna.

4.2 Funkce aplikace

Jelikož se jedná o aplikaci zaměřenou na správu překladu, je zde kladen velký důraz na práci s textem a uživatelskou interakci v rámci překladů. Samotná aplikace dovoluje uživateli vytvářet projekty pro překlady, ve kterých určí výchozí jazyk všech překladů a jazyky do kterých se texty budou překládat. Překlady však nemusí tvořit sám, je mu umožněno pracovat na projektech ve více lidech. Díky tomu mohou být překlady upravovány více uživateli, kteří mají v rámci projektu nastavená různá uživatelská práva a v tuto chvíli může být uživatel v rámci projektu jako **administrátor** nebo **překladatel**. Administrátor má možnost spravovat uživatele a zároveň upravovat překlady, překladatel může jen upravovat překlady.

V rámci práce s překlady jsou uživatelům nabídnuty funkce, které jim mohou usnadnit překlad vytvořit nebo upravit. Mezi hlavní takové funkce slouží strojový překlad, díky kterému si může uživatel nechat přeložit celý zdrojový text do překládaného jazyka. Dále je uživateli nabídnuta možnost kontroly pravopisu s možností nabídky opravy těchto slov. Pro lepší orientaci v překladech může uživatel vidět poslední změnu v překladech a historii těchto překladů.

V rámci tabulky všech překladů může uživatel pro lepší orientaci využít filtr s možností zobrazení chybějících překladů, nebo vyhledání přímo přeloženého slova nebo fráze. Uživatel má rovněž možnost nahrát nebo stáhnout soubor s překlady.

4.2.1 Autorizace uživatele

Pro aplikaci je důležité, aby nebyla přístupná pro kohokoli. Uživatelé mají k dispozici pouze svá data a ne data jiných uživatelů. K tomu nám dopomáhá autorizace uživatelů, díky ní se každý uživatel dostane pouze ke svým datům, které sám upravuje a nemůže díky tomu dojít ke ztrátě nebo poškození cizích dat. Také to zabraňuje využití aplikace komukoli, kdo není registrovaný a přihlášený. Díky tomu může být větší přehled o tom, co který uživatel v aplikaci dělal.

Autorizace uživatele je řešena pomocí JWT tokenu, který se vytváří při přihlášení uživatele. Na straně serveru se pomocí *JwtSecurityTokenHandler* vytvoří token, který se následně posílá v hlavičce všech autorizovaných dotazů a na základě kterého je kontrolován přihlášený uživatel. Při vypršení tohoto tokenu je po uživateli vyžadováno opětovné přihlášeni.

```
private string GenerateJwtToken(User user)
{
   var tokenHandler = new JwtSecurityTokenHandler();
   var key = Encoding.ASCII.GetBytes(_appSettings.Secret);
   var tokenDescriptor = new SecurityTokenDescriptor
   {
      Subject = new ClaimsIdentity(new Claim[]
      {
```

```
            new Claim(ClaimTypes.Name, user.Id.ToString())
        }),
        Expires = DateTime.UtcNow.AddDays(7),
        SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key),
            SecurityAlgorithms.HmacSha256Signature)
    };
    var token = tokenHandler.CreateToken(tokenDescriptor);
    return tokenHandler.WriteToken(token);
}
```

Listing 4.1: Metoda generující JWT token pro autorizaci

4.2.2 Strojový překlad

Strojový překlad je funkce, která uživateli usnadní práci a je základní funkcí při práci s překlady. Jedná se o rychlý způsob jak jednoduše dostat minimálně povědomí o překládaném textu. Strojové překlady jsou v dnešní době na dobré úrovni, avšak stále nejsou stoprocentní, proto se na ně nemůžeme plně spoléhat. Vytvořený systém dovoluje uživateli strojový překlad použít a následně jej dále upravovat tak, aby co nejlépe vystihl daný kontext textu.

Strojový překlad je v aplikaci řešen pomocí napojení na API třetí strany. Přesněji bylo využito překladače od společnosti Google, kde pomocí rest API dotazů je posílán request přímo na překladač, který vrací zpět přeložené data. Tato komunikace je řešena na úrovni webové aplikace, a je posílaná pomocí knihovny *Axios*.

```
const getTranslation = async () => {
    const sourceLanguage = languages[0].code;
     const targetLanguage = languages.find((e) => e.id === selectedLang)?.code;
     const sourceTranslation = editedTranslations.find(e => e.langId === languages
         [0].id)?.translate;

    if (sourceLanguage !== targetLanguage) {
        const res = await axios.get(
            process.env.REACT_APP_GOOGLE_TRANSLATE_URL || "https://translate.
                googleapis.com/translate_a/single",
            {
                params: {
                    client: "gtx",
                    dt: "t",
                    q: sourceTranslation,
```

```
            tl: targetLanguage,
            sl: sourceLanguage,
          },
        }
      );

      setAutoTranslate(res.data?.[0]?.[0]?.[0] || "");
    }
};
```

Listing 4.2: Funkce posílající dotaz na API Google překladače

Uživatel má poté možnost požít překlad. Ten následně může upravovat tak, aby byl překlad co nejlépe použitelný pro daný tvar.

4.2.3 Kontrola pravopisu s možností opravy

Kontrola pravopisu a následná oprava je funkce, která je v dnešní době nutná při práci s jakýmkoli textem. Kontrola textu v této aplikaci je postavena na jednoduchém principu vyhledávání slov v existujících slovnících pro daný jazyk. I tato kontrola je nám schopna pomoci odstranit drobné pravopisné chyby či překlepy v textu. Společně s možností automatické opravy je možno uživateli zobrazit návrhy, které by mohli nahradit špatně napsané slovo či gramaticky nesprávné.

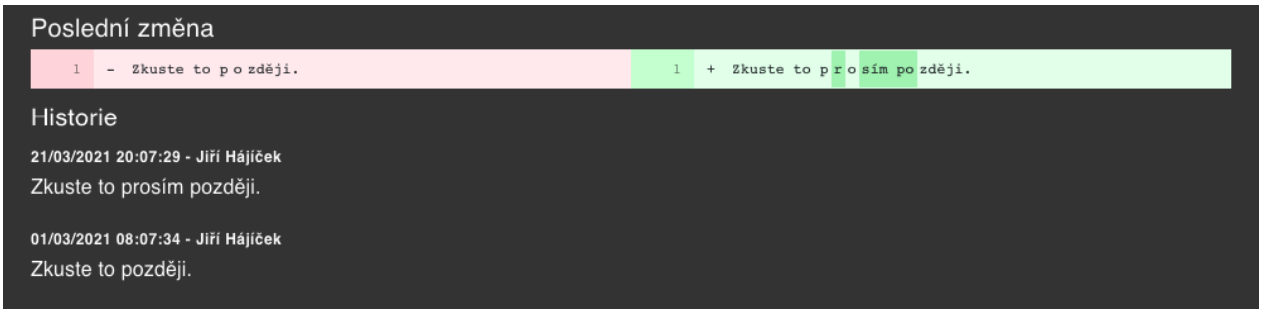
Kontrola pravopisu je řešena na severu pomocí knihovny *Hunspell*, kde metoda pro kontrolu textu kontroluje slovo po slově zda existuje v knihovně pro daný jazyk. Pokud toto slovo nenalezne, uživateli se ve webové aplikaci zobrazí jako podtržené. Při kliknutí na toto slovo se zobrazí dialog s nabídkou možných slov, kterými je může uživatel nahradit.

Kontrola pravopisu s návrhem opravy je řešena pomocí veřejného Bing API, kterého se aplikace dotazuje. Toto API je v současné době omezeno na 1000 dotazů, neboť se jedná o zkušební verzi, při ostrém provozu aplikace by bylo nutné tento limit navýšit.

4.2.4 Historie překladu

Historie překladu umožňuje uživateli zobrazit historické verze překladů, a to společně s informací kdy byly vytvořeny a kdo tento překlad vytvořil. Jakoukoli následnou editací překladu vzniká nová verze, kdy původní verze se zobrazí v historii překladu. Uživatel má také možnost podívat se na poslední úpravu překladu a může vidět vše, co vše bylo změno, odstraněno nebo přidáno.

Historie překladu je řešena hlavně na úrovni databáze, kde při každé změně překladu vzniká nový záznam v tabulce *Translations*. Jediné co se u starého překladu změní je atribut **active**, který se změní na *false*. Jediný aktivní překlad tak zůstává poslední vytvořený, a pro zobrazení historie, se tak pouze zobrazují záznamy, které mají schodný klíč, projekt, jazyk a nejsou aktivní.



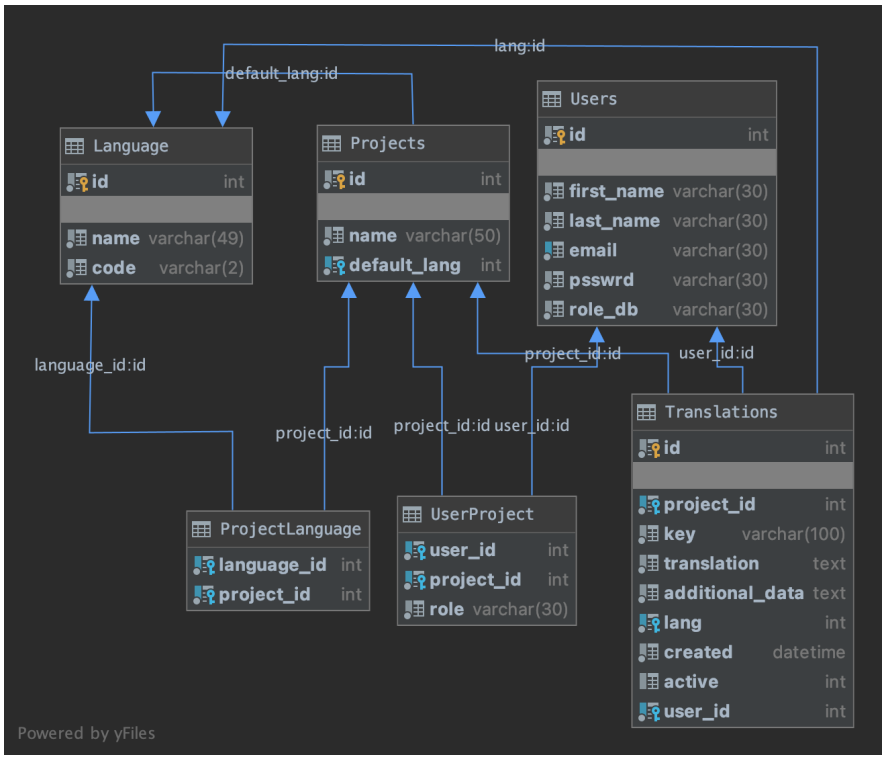
Obrázek 4.1: Historie překladu

4.3 Implementace webové aplikace

Celý systém je postaven na třívrstvé architektuře, kde dochází ke spojení databáze, serveru a samotné webové aplikace. Každá vrstva má na starost jinou práci v rámci aplikace. Datová vrstva je zastoupena databází která dlouhodobě ukládá data uživatelů. Aplikační vrstva v tomto případě slouží hlavně pro komunikace mezi prezenční a datovou vrstvou. Toto spojení je řešeno pomocí .NET aplikace. Prezenční vrstva zobrazuje data uživateli a řeší logiku zpracování formulářů a některých dat. Tato vrstva je tvořena React aplikací, která je spouštěna v internetovém prohlížeči.

4.3.1 Databáze

Samotná databáze je tvořena několika tabulkami, které ukládají data uživatelů. Tyto tabulky jsou na sebe napojeny tak, aby bylo dosaženo efektivního uložení dat. Hlavní a největší tabulka je tabulka *Translations*, ve které jsou uloženy veškeré překlady a na tuto tabulku jsou napojeny další tabulky. Díky těmto tabulkám je jasné do kterého projektu překlad patří a který uživatel překlad vytvořil a také v jakém jazyce je překlad napsán.



Obrázek 4.2: Návrh databázového modelu

Velká část logiky pro získávání dat je řešena o vrstvu výše, tedy v aplikační vrstvě. V tomto případě v .NET aplikaci. Zde je využito *Entity Frameworku* pro snazší komunikaci s databázi.

4.3.2 .NET aplikace

V případě této aplikace slouží .NET aplikace hlavně jako spojení mezi prezenční vrstvou (React aplikací) a datovou vrstvou (MySQL databází), avšak se zde nachází i vlastní vytvořené funkce. Celá aplikace je postavena na frameworku APS.NET Core, který je vyvíjený společností Microsoft. Jedná se o framework, který vytváří webové rozhraní API, díky kterému komunikuje prezenční vrstva s aplikační. Jednotlivé API rozhraní jsou popsány v kontrolerech, které jsou napojeny na jednotlivé servisy. V těch je řešena komunikace s databází pomocí Entity Frameworku nebo je zde řešena funkční logika. Jedna z vytvořených logických funkcí je funkce, která kontroluje správnost pravopisu.

```
public List<CheckSpell> CheckSpell(string text, string lang)
{
   List<CheckSpell> checkSpell = new List<CheckSpell>();

   var dictionary = WordList.CreateFromFiles(@"en_US.dic");
```

```
    if (lang == "cs")
    {
        dictionary = WordList.CreateFromFiles(@"cs_CZ.dic");
    }
    ...

    String[] splited = text.Split(' ');

    for (int i = 0; i < splited.Length; i += 1)
    {
        if (!splited[i].Equals(""))
        {
            CheckSpell spell = new CheckSpell();
            spell.Found = dictionary.Check(splited[i]);
            spell.Word = splited[i];
            checkSpell.Add(spell);
        }
    }

    return checkSpell;
}
```

Listing 4.3: Metoda na kontrolu pravopisu

Další metody slouží už jen pro zápis nebo čtení z databáze. Většina další aplikační logiky je řešena na prezenční vrstvě aplikace, tedy v samotné React aplikaci.

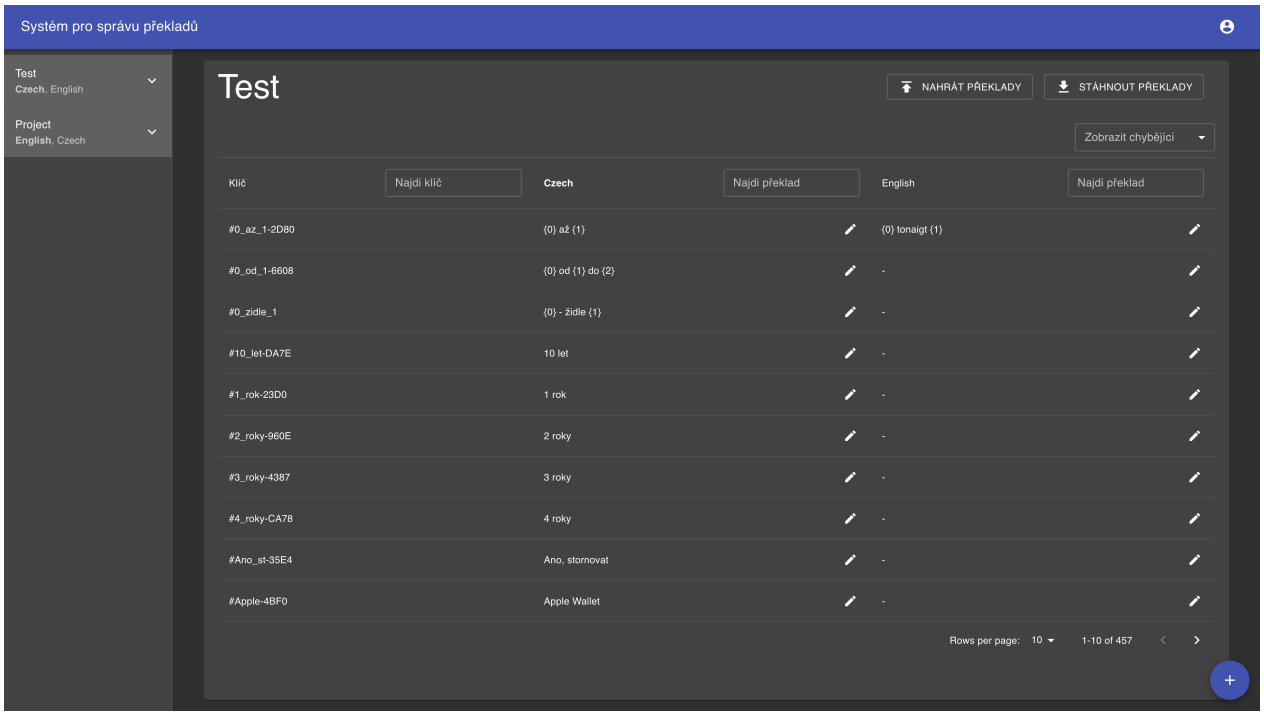
4.3.3 Webová React aplikace

React aplikace tvoří prezenční vrstvu, ve které je částečně řešena logika některých funkčních částí. Základem celé aplikace je *React router*, který se stará o routování celé aplikace a díky němuž se ze single-page aplikace stane aplikace s více stránkami. Jednotlivé stránky jsou pak uloženy v samostatných souborech, kde je popsána jejich logika a jejich vzhled, který je často tvořen dalšími samostatnými komponentami.

Samotné komponenty tvoření různé dialogy a menší části aplikace. Pro lepší uživatelské prostředí byla využita knihovna *Material-UI*, která obsahuje React komponenty představující různá tlačítka, dialogy, inputy a spousty dalších funkčních komponent. Díky této knihovně dostává aplikace jednotný design. Pro uživatelé jsou mnohdy komponenty z této knihovny dobře známé, jelikož jsou hojně využívány společností Googole. V komponentách je navíc řešeno i samotné manipulování

s daty. Ať už se jedná o data uživatele, která se posílají na backend, nebo data, která jsou z backendu přijímána a zobrazována uživateli. Při práci s daty je často řešena validace dat posílaných na backend nebo filtrování zobrazovaných překladů v tabulce.

Pro lepší manipulování komunikace s .NET aplikací, byla vytvořena funkce pro odesílání dotazů. Tato funkce zároveň řeší některé chyby, které v komunikaci mohou nastat. Jedná o moment, kdy přijde odpověď na dotaz, že uživatel není autorizován a v tento okamžik je uživatel automaticky odhlášen a je po něm vyžadováno opětovné přihlášení. Tato logika by však nebyla možná bez použití React Context, který složí ke sdílení dat napříč komponenty. Díky tomu tak jedna komponenta může ovlivnit chovaní jiné komponenty, úplně na druhé straně aplikace.



Obrázek 4.3: Uživatelské rozhraní aplikace

Každý projekt pro překlady je pak rozdělen do dvou samostatných stránek. Jedna řeší informace o projektu a uživatele, kteří jsou do projektu zapojeni. Druhá stránka řeší překlady samotné. Na stránce s překlady je zobrazena tabulka s překlady, které jsou rozděleny na stránky, a ve kterých může uživatel filtrovat nebo procházet samotnými stránkami s překlady. Tato logika je řešena v rámci prezenční vrstvy v React aplikaci. Při editaci překladu je uživateli nabídnuta možnost zobrazit si strojový překlad, jsou znázorněny chybně napsaná slova a historie překladů. Další stránky aplikace jsou nastavení účtu uživatele a přihlašovací a registrační stránka.

Kapitola 5

Závěr

V první části bylo popsáno několik systému pro správu překladů, a to systém Lokalise, Phrase a český systém Localazy. Každý z těchto systému byl odzkoušen a byly popsány jednotlivé jejich vlastnosti a funkce, které tyto systémy nabízí uživatelům.

V další části práce bylo představeno několik moderních řešení pro tvorbu webových aplikací. Jednalo se o JavaScriptové knihovny nebo frameworky. Byly představeny systém React, Vue.JS, Angular a Ember a důkladně popsány jejich vlastnosti, funkční závislosti a také výhody, které nabízí. U každého tohoto systému byla popsána práce s DOM a způsob jak se tvoří komponenty a pracuje s vývojem celého sytému.

Začátek poslední kapitoly se věnuje popisu použitých technologií při implementaci systému. Jedná se o technologie MySQL, použité pro databáze a technologii .NET, která se stará se o komunikaci mezi databázi a samotnou webovou aplikací, pro kterou byla použita knihovna React. V další části této kapitoly byly popsány klíčové funkce systému pomáhající uživatelům při práci s překlady. Poslední část kapitoly byla zaměřena na celkovou konstrukci systému a samotnou implementaci.

Literatura

1. *Lokalise app* [online] [cit. 2021-04-01]. Dostupné z: `https://lokalise.com/`.
2. *Phrase app* [online] [cit. 2021-04-01]. Dostupné z: `https://phrase.com/`.
3. *Localazy app* [online] [cit. 2021-04-01]. Dostupné z: `https://localazy.com/`.
4. *Documentation for PHP* [online] [cit. 2021-04-01]. Dostupné z: `https://www.php.net/docs.php`.
5. ŽÁRA, Ondřej. *JavaScript Programátorské techniky a webové technologie*. Computer Press, 2016.
6. CHINNATHAMBI, Kirupa. *Learning React*. Pearson Education (US), 2018.
7. FILIPOVA, Olga. *Learning Vue.js 2*. Packt Publishing Limited, 2016.
8. FREEMAN, Adam. *Pro Angular 9*. APress, 2020.
9. *Ember.js Guides* [online] [cit. 2021-04-01]. Dostupné z: `https://guides.emberjs.com/`.
10. *Wikipedia - MySQL* [online] [cit. 2021-04-01]. Dostupné z: `https://en.wikipedia.org/wiki/MySQL`.
11. FREEMAN, Adam. *Pro ASP.NET Core 3*. APress, 2020.